Internet Programming in Python - Week 7

Large web frameworks - Django

http://en.wikipedia.org/wiki/File:Django_Reinhardt_(Gottlieb_07301).jpg

http://djangopony.com/

Brian Dorsey
brian@dorseys.org

ANNOUNCEMENTS

Portland Python sprint
Saturday, February 26th

http://goo.gl/Ln68F

http://www.djangocon.eu/

class field-trip?

Python 3.2 final released!

http://www.python.org/
download/releases/3.2/

http://docs.python.org/
dev/whatsnew/3.2.html

a note on the format today

a bit higher level than previous classes, both because we have two classes on Django, and it's just really big.

A moment to reflect

not just a moment, today

Questions and review

(20)

some thoughts from the assignments

turn in here: `http://is.gd/uwipip_week6`

Assignment

- week 6 assignment == week 7 reading
- do the Django tutorial (part 1, 2, 3): `http://docs.djangoproject.com/en/1.2/intro/tutorial01/`
- work through it locally, later push to VM
  ok to use development server:
  `python manage.py runserver 0.0.0.0:8000`
- stretch goal: also add and document a REST/JSON API for poll results

how'd it go?

only got 15 turned in

most worked great

TemplateDoesNotExist at /polls/2/

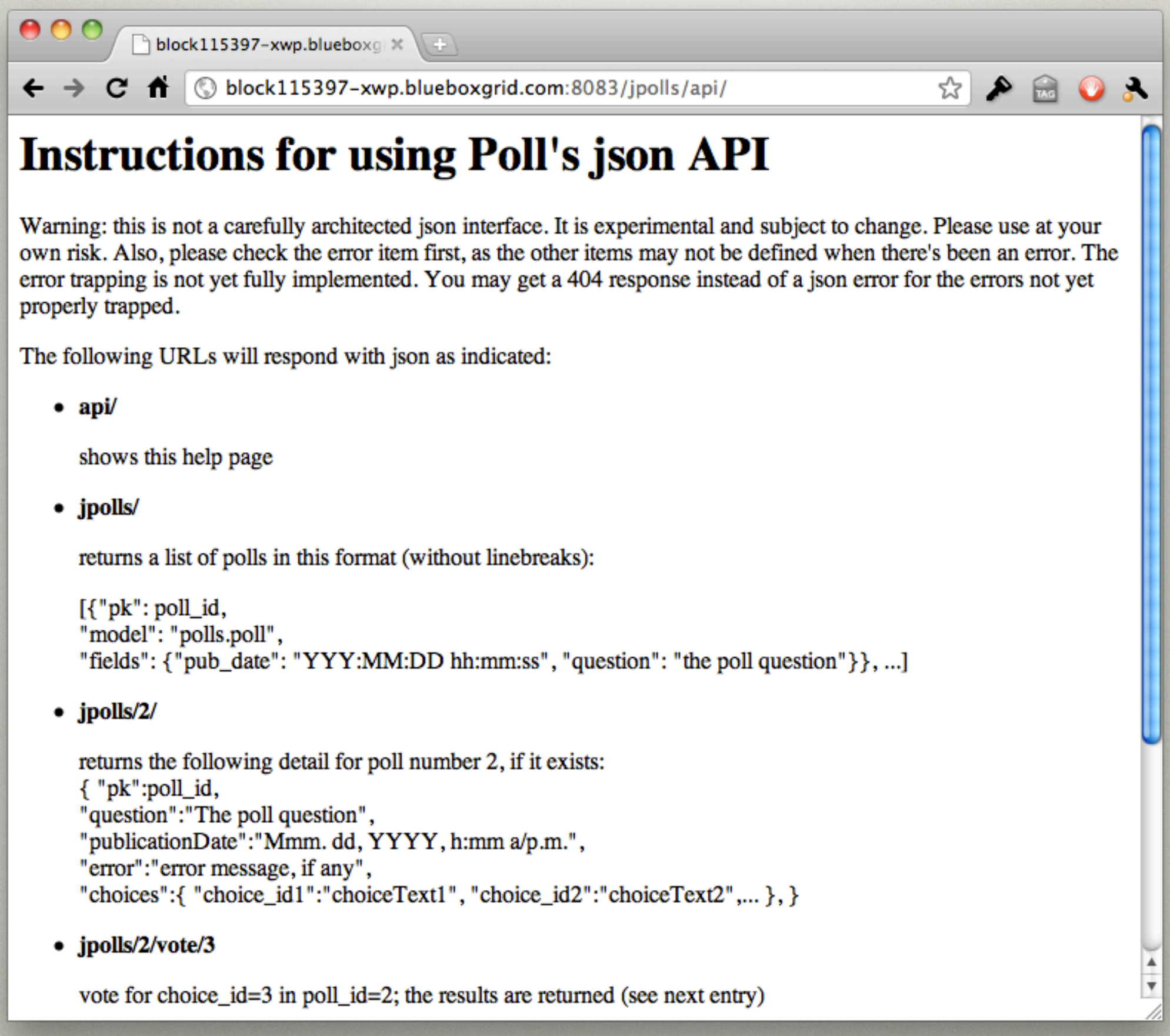
polls/detail.html

Request Method:	GET
Request URL:	http://block115402-4qe.blueboxgrid.com:8000/polls/2/
Django Version:	1.2.5
Exception Type:	TemplateDoesNotExist
Exception Value:	`polls/detail.html`
Exception Location:	/usr/local/lib/python2.6/dist-packages/django/template/loader.py in fin
Python Executable:	/usr/bin/python
Python Version:	2.6.5
Python Path:	['/home/tothg/PythonClass/Week6Assignment/mysite', '/usr/lib/python '/usr/lib/python2.6/plat-linux2', '/usr/lib/python2.6/lib-tk', '/usr/lib/python '/usr/lib/python2.6/lib-dynload', '/usr/lib/python2.6/dist-packages', '/usr/lib/pymodules/python2.6', '/usr/local/lib/python2.6/dist-packages'
Server time:	Tue, 22 Feb 2011 18:46:44 -0600


Log in | Django site admin
block115393-ake.blueboxgrid.com/admin/
Django administration
Username:
Password:
Log in

CherryPy?!?!? :)


Login | Django site admin
block115393-ake.blueboxgrid.com/admin/
Django administration
Username:
Password:
Log in


block115397-xwp.blueboxg
block115397-xwp.blueboxgrid.com:8083/jpolls/api/
Instructions for using Poll's json API
Warning: this is not a carefully architected json interface. It is experimental and subject to change. Please use at your own risk. Also, please check the error item first, as the other items may not be defined when there's been an error. The error trapping is not yet fully implemented. You may get a 404 response instead of a json error for the errors not yet properly trapped.
The following URLs will respond with json as indicated:
api/
shows this help page
jpolls/
returns a list of polls in this format (without linebreaks):
[{"pk": poll_id,
"model": "polls.poll",
"fields": {"pub_date": "YYY:MM:DD hh:mm:ss", "question": "the poll question"}}, ...]
jpolls/2/
returns the following detail for poll number 2, if it exists:
{ "pk":poll_id,
"question":"The poll question",
"publicationDate":"Mmm. dd, YYYY, h:mm a/p.m.",
"error":"error message, if any",
"choices":{ "choice_id1":"choiceText1", "choice_id2":"choiceText2",... }, }
jpolls/2/vote/3
vote for choice_id=3 in poll_id=2; the results are returned (see next entry)

“i was able to use my iPhone to install, port and troubleshoot the django app from work (firewall, no ssh, etc) using TouchTerm app. Just git pulled from github to the VM. Awesome!”

gotchas

permissions
(as always)

```
'string' 'concatenation'
'two', 'strings'
```

assumptions in the docs

global 404s

vs.

view specific 404s

browser caches
hiding changes

secondary objective
or
what *is* the best way
to get JSON output?

don’t go through an HTML template

roll your own?

```
from polls.models import Poll
import json

p = Poll.objects.get(pk=1)
data = dict(question = p.question,
            pub_date = p.pub_date.isoformat())
choices = []
for choice in p.choice_set.all():
    choices.append(dict(choice = choice.choice,
                        votes=choice.votes))

data['choices'] = choices
print json.dumps(data, indent=4)
```


get some lists and dictionaries and pass them to simplejson.dumps()

roll your own?

```
{
    "pub_date": "2011-02-19T12:02:55.397075",
    "question": "What's up?",
    "choices": [
        {
            "votes": 0,
            "choice": "not much"
        },
        {
            "votes": 0,
            "choice": "the sky"
        }
    ]
}
```

probably ought to include IDs of some sort if you ever want to do updates...

django.core.serializers?

```
print serializers.serialize('json', Poll.objects.all
(), indent=4)
[
    {
        "pk": 1,
        "model": "polls.poll",
        "fields": {
            "pub_date": "2011-02-19 12:02:55",
            "question": "What's up?"
        }
    }
]
```

http://docs.djangoproject.com/en/1.2/topics/serialization/

The Git Parable

http://tom.preston-werner.com/2009/05/19/the-git-parable.html

did you read it? did it make sense?

Lecture A (25)

full-stack frameworks landscape

common:

open source

mature

Python. ;)

Most of the web frameworks have a BSD style license, like Python.

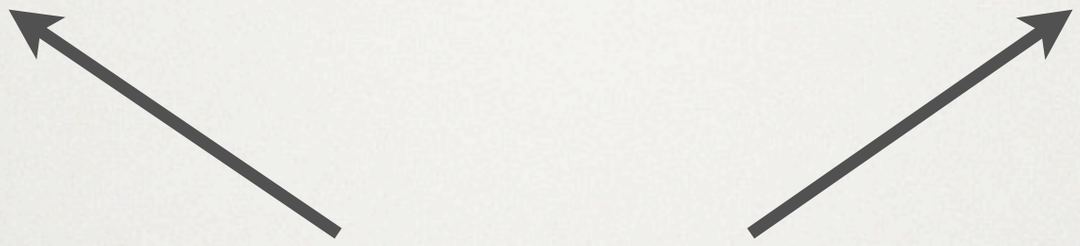
Zope

http://www.zope.org/

The first full-stack Python web framework. Possibly the first web framework. Even includes it's own object oriented database. Since at least 1999. ZODB, templates, through the web editing & config.

Grok

Plone



Zope

http://www.zope.org/

Grok

Zope

http://grok.zope.org/

What is Grok?

Grok is a web application framework for Python developers. It is aimed at both beginners and very experienced web developers. Grok has an emphasis on agile development. Grok is easy and powerful.

You will likely have heard about many different web frameworks for Python as well as other languages. Why should you consider Grok?

- Grok offers a *lot* of building blocks for your web application.
- Grok is informed by a *lot* of hard-earned wisdom.

Grok accomplishes this by using at its core the Zope Toolkit (ZTK), an advanced object-oriented set of libraries intended for reuse by web frameworks. While Grok uses the Zope Toolkit, and benefits a lot from it, you can get started with Grok without any special knowledge of the ZTK.

Plone

Zope

http://seattleplone.org/ http://plone.org/about/ http://plone.org/

Huge project. Content Management System. Enterprise Open Source. Build intranets & the public face of companies. Out of the box internationalized and compliant with many, many standards and regulations. Compare with Sharepoint & Drupal to some degree.

What is Plone?

by Alexander Limi — last modified Sep 08, 2010 05:24 AM

A powerful, flexible Content Management Solution that is easy to install, use and extend.

Plone lets non-technical people create and maintain information for a public website or an intranet using only a web browser. Plone is easy to understand and use—allowing users to be productive in just half an hour—yet offers a wealth of community-developed add-ons and extensibility to keep meeting your needs for years to come.

Blending the creativity and speed of open source with a technologically advanced Python back-end, Plone offers superior security without sacrificing power or extensibility.

The Plone community is an incredibly diverse group that bridges many types and sizes of organizations, many countries and languages, and everything from technical novices to hardcore programmers. Out of that diversity comes an attention to detail in code, function, user interface and ease of use that makes Plone one of the top 2% of open source projects worldwide. (Source: Ohloh)

Plone's intellectual property and trademarks are protected by the non-profit Plone Foundation. This means that Plone's future is not in the hands of any one person or company.

Thousands of websites across large and small businesses, education, government, non-profits, sciences, media and publishing are using Plone to solve their content management needs. This is supported by a global network of over 300 solution providers in more than 50 countries. Looking for a hosting provider to host your Plone site for you? You can find a list of providers and consultants on plone.net.

We are very proud to be known by the company we keep. Organizations as diverse as NASA, Oxfam, Amnesty International, Nokia, eBay, Novell, the State Universities of Pennsylvania and Utah, as well as the Brazilian and New Zealand governments— all use Plone.

web2py

http://www.web2py.com/

Looks pretty amazing. We're not going to cover it at all. :(
Each of us should probably do a project in web2py, just to see what it's like.

Why web2py?

- **Created by a community of professionals** and University professors in Computer Science and Software Engineering.
- **Always backward compatible.** We have not broken backward compatibility since version 1.0 in 2007, and we pledge not to break it in the future.
- **Easy to run.** It requires no installation and no configuration.
- **Runs on** Windows, Mac, Unix/Linux, Google App Engine, Amazon EC2, and almost any web hosting via Python 2.4/2.5/2.6/2.7, or Java with Jython.
- **Runs with** Apache, Lighttpd, Cherokee and almost any other web server via CGI, FastCGI, WSGI, mod_proxy, and/or mod_python. It can embed third party WSGI apps and middleware.
- **Talks to** SQLite, PostgreSQL, MySQL, MSSQL, FireBird, Oracle, IBM DB2, Informix, Ingres, and Google App Engine.
- **Secure** It prevents the most common types of vulnerabilities including Cross Site Scripting, Injection Flaws, and Malicious File Execution.
- **Enforces good Software Engineering practices** (Model-View-Controller design, Server-side form validation, postbacks) that make the code more readable, scalable, and maintainable.
- **Speaks multiple protocols** HTML/XML, RSS/ATOM, RTF, PDF, JSON, AJAX, XML-RPC, CSV, REST, WIKI, Flash/AMF, and Linked Data (RDF).
- **Includes** a SSL-enabled and streaming-capable web server, a relational database, a web-based integrated development environment and web-based management interface, a Database Abstraction Layer that writes SQL for you in real time, internationalization support, multiple authentication methods, role based access control, an error logging and ticketing system, multiple caching methods for scalability, the jQuery library for AJAX and effects. Read more...

Pyramid = Pylons + repoze.bfg

http://pylonsproject.org/

This is where to go if you want a bigger framework and flexibility.
(projects joined recently)

Why use Pylons?

Pylons combines the very best ideas from the worlds of Ruby, Python and Perl, providing a structured but extremely flexible Python web framework. It's also one of the first projects to leverage the emerging WSGI standard, which allows extensive re-use and flexibility — but only if you need it. Out of the box, Pylons aims to make web development fast, flexible and easy. Find out more, install the latest version, or **read the new Pylons book**.

Who uses Pylons?

A few of the many companies and websites that use Pylons:

Disclaimer: These companies do not necessarily run their public websites with Pylons, nor officially endorse Pylons. All logos Trademark of the respective company.

See more companies/sites using Pylons

Plays Well With Others

Pylons is built on Paste and allows and encourages use of your favorite Python components and libraries:

- Models: SQLAlchemy, SQLObject, CouchDB, or none at all
- Templating: Mako, Genshi, Jinja2, or whatever you like
- Helpers: WebHelpers for small HTML snippets, FormAlchemy generates entire forms
- Request Dispatching: Routes by default, or plug in your favorite

Not sure which one to choose? No problem! The Pylons documentation recommends a powerful templating engine (Mako) and database ORM (SQLAlchemy) to help you get started.

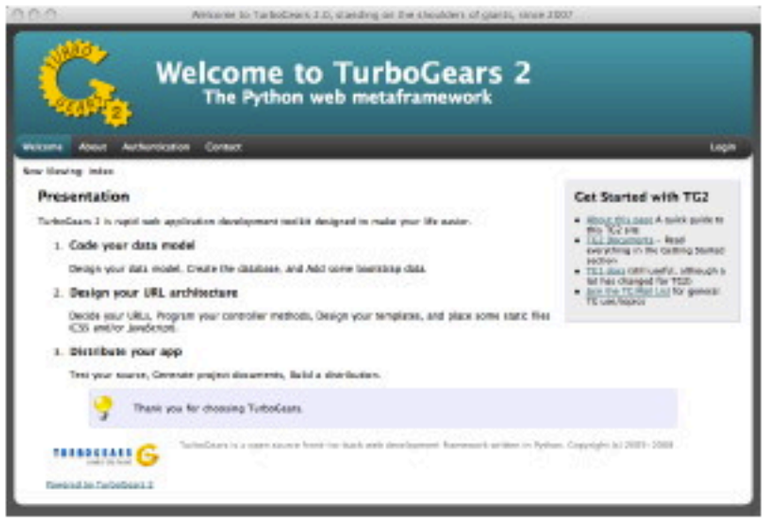
Turbogears

Pyramid

http://www.turbogears.org/

Build a database-driven app in minutes!

Create a database-driven, ready-to-extend application in minutes. All with designer friendly templates, easy AJAX on the browser side *and* on the server side, with an incredibly powerful and flexible Object Relational Mapper (ORM), and with code that is as natural as writing a function.



Get started Learning TurboGears 2 by looking at our famous wiki tutorial.

A new Generation in dynamic web frameworks

TurboGears 2 is the built on top of the experience of several next generation web frameworks including TurboGears 1 (of course), Django, and Rails. All of these frameworks had limitations which were frustrating in various ways, and TG2 is an answer to that frustration. We wanted something that had:

- Real multi-database support
- Horizontal data partitioning (sharding)
- Support for a variety of JavaScript toolkits, and new widget system to make building ajax heavy apps easier
- Support for multiple data-exchange formats.
- Built in extensibility via standard WSGI components

Django

http://www.djangoproject.com/

This is where we'll be spending the next two weeks.

Meet Django

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design.

Developed four years ago by a fast-moving online-news operation, Django was designed to handle two challenges: the intensive deadlines of a newsroom and the stringent requirements of the experienced Web developers who wrote it. It lets you build high-performing, elegant Web applications quickly.

Django focuses on automating as much as possible and adhering to the DRY principle.

Dive in by reading the overview →

When you're ready to code, read the installation guide and tutorial.

The Django framework

Object-relational mapper

Define your data models entirely in Python. You get a rich, dynamic database-access API for free — but you can still write SQL if needed.

Automatic admin interface

Save yourself the tedious work of creating interfaces for people to add and update content. Django does that automatically, and it's production-ready.

Elegant URL design

Design pretty, cruft-free URLs with no framework-specific limitations. Be as flexible as you like.

Template system

Use Django's powerful, extensible and designer-friendly template language to separate design, content and Python code.

Cache system

Hook into memcached or other cache frameworks for super performance — caching is as granular as you need.

Internationalization

Django has full support for multi-language applications, letting you specify translation strings and providing hooks for language-specific functionality.

recommendation?

Any guesses?

"The web framework for perfectionists with deadlines."

http://www.djangoproject.com/

what is it?

the web framework Goldilocks would choose

seems to hit a sweet spot - does almost all the repetitive stuff, but still allows you to customize easily

(for building a content oriented website,

or when internal people will edit the data)

Meet Django

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design.

Developed four years ago by a fast-moving online-news operation, Django was designed to handle two challenges: the intensive deadlines of a newsroom and the stringent requirements of the experienced Web developers who wrote it. It lets you build high-performing, elegant Web applications quickly.

Django focuses on automating as much as possible and adhering to the DRY principle.

Dive in by reading the overview →

When you're ready to code, read the installation guide and tutorial.

The Django framework

Object-relational mapper

Define your data models entirely in Python. You get a rich, dynamic database-access API for free — but you can still write SQL if needed.

Automatic admin interface

Save yourself the tedious work of creating interfaces for people to add and update content. Django does that automatically, and it's production-ready.

Elegant URL design

Design pretty, cruft-free URLs with no framework-specific limitations. Be as flexible as you like.

Template system

Use Django's powerful, extensible and designer-friendly template language to separate design, content and Python code.

Cache system

Hook into memcached or other cache frameworks for super performance — caching is as granular as you need.

Internationalization

Django has full support for multi-language applications, letting you specify translation strings and providing hooks for language-specific functionality.

Good stuff... but a lot of it is pretty common these days.

Meet Django

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design.

Developed four years ago by a fast-moving online-news operation, Django was designed to handle two challenges: the intensive deadlines of a newsroom and the stringent requirements of the experienced Web developers who wrote it. It lets you build high-performing, elegant Web applications quickly.

Django focuses on automating as much as possible and adhering to the DRY principle.

Dive in by reading the overview →

When you're ready to code, read the installation guide and tutorial.

The Django framework

Object-relational mapper

Define your data models entirely in Python. You get a rich, dynamic database-access API for free — but you can still write SQL if needed.

Automatic admin interface

Save yourself the tedious work of creating interfaces for people to add and update content. Django does that automatically, and it's production-ready.

Elegant URL design

Design pretty, cruft-free URLs with no framework-specific limitations. Be as flexible as you like.

Template system

Use Django's powerful, extensible and designer-friendly template language to separate design, content and Python code.

Cache system

Hook into memcached or other cache frameworks for super performance — caching is as granular as you need.

Internationalization

Django has full support for multi-language applications, letting you specify translation strings and providing hooks for language-specific functionality.

Object-relational mapper

Define your data models entirely in Python. You get a rich, dynamic database-access API for free — but you can still write SQL if needed.

Automatic admin interface

Save yourself the tedious work of creating interfaces for people to add and update content. Django does that automatically, and it's production-ready.

very active

docs
examples
apps
jobs

documentation

We've put a lot of effort into making Django's documentation useful, easy to read and as complete as possible. The rest of this document explains more about how the documentation works so that you can get the most out of it.

(Yes, this is documentation about documentation. Rest assured we have no plans to write a document about how to read the document about documentation.)

Finding documentation

Django's got a *lot* of documentation – almost 200,000 words – so finding what you need can sometimes be tricky. A few good places to start are the Search Page and the Index.

http://docs.djangoproject.com/en/1.2/intro/whatsnext/

principles

Straight from http://toys.jacobian.org/presentations/2008/pycon/tutorial/

“Django is a high-level Python web framework that encourages rapid development and clean pragmatic design.”

Straight from http://toys.jacobian.org/presentations/2008/pycon/tutorial/

From: http://toys.jacobian.org/presentations/2008/pycon/tutorial/
“This is the one sentence introduction to Django; the “mission statement” as it were. Let me break it down.”

“Django is a high-level Python web framework...”

Straight from http://toys.jacobian.org/presentations/2008/pycon/tutorial/

abstracts common problems, provides shortcuts
smooth over pain points, but stay out of the way
never worry about HTTP, SQL... unless you want to
Python itself is a feature, it makes a point about using real Python, not inventing a sub-language

From: http://toys.jacobian.org/presentations/2008/pycon/tutorial/
“A Web framework is software that abstracts common problems of Web development and provides shortcuts for frequent programming tasks.
A good web framework finds the “pain points” of web developers and smoothes them over — but never gets in the way! It should let you work at a much higher level of abstraction, so you don’t need to worry about details of HTTP, SQL, or whatever. Again, it shouldn’t get in your way if you need to “step down” a level.
Python itself is another key “feature” of Django. It’s a beautiful, concise, powerful, high-level language with an amazing community; many things are easier in Django simply because of the tools you can build on top of.
Django also makes a point of not changing anything about how Python works; if you learn Django, you’re also learning Python. This helps down the road.”

“...that encourages
rapid development...”

Straight from http://toys.jacobian.org/presentations/2008/pycon/tutorial/

worthless if it doesn’t save time
build websites in hours or days, not weeks or years
born from real problems - every convenience is there because it makes you more productive

From: http://toys.jacobian.org/presentations/2008/pycon/tutorial/
“Regardless of how many powerful features it has, a Web framework is worthless if it doesn't save you time. Django's philosophy is to do all it can to facilitate hyper-fast development. With Django, you build Web sites in a matter of hours, not days; weeks, not years.
This comes directly out of real-world problems. We’re programmers, yes, but we work at a news organization. When a big story breaks, we don’t have the luxury of a long development cycle. Every convenience in Django is there because it makes you more productive.”

“...and clean,
pragmatic design.”

Straight from http://toys.jacobian.org/presentations/2008/pycon/tutorial/

From: http://toys.jacobian.org/presentations/2008/pycon/tutorial/
“Django strictly maintains a clean design throughout its own code and makes it easy to follow best Web-development practices in the applications you create.
The philosophy here is to make it easy to do things the "right" way.”

design philosophy

http://docs.djangoproject.com/en/1.2/misc/design-philosophies/

history

short version

experienced devs at a small newspaper made a great tool and convinced their bosses to open-source it

more details:

http://www.quora.com/What-is-the-history-of-the-Django-web-framework

a list of rejected names for the framework that become Django:

http://jacobian.org/writing/private_dancer/

brazos
superglue
bodhisattva
webbing
boidae
bohdi
skoro
consolidata
piston
physique
silhouette
valance
anson
The Python Web Framework (PWF)

Tornado
Publishing System
Web
Type Framework
Pypeline
super
magic machine
magic machine
private dancer
“the CMS”
The Give-a-Damn machine
boltdozer
banister
garbonzo
Simon

fizgig
palmy
cogent
pith
pithy
pyth
pythy
festoon
poeks
lavalier
clerisy
bandwidth
django

http://jacobian.org/writing/private_dancer/

Lab A

(20)

Lab A

- make groups of four-ish people
- You're now a Committee.

  Here is your task:
  `http://is.gd/uwipip_week7_lab`

solutions

Break

(10)

API Talks (20)

talks

Guest Speaker

(30)

Gary Bernhardt

http://blog.extracheese.org/

Break

(10)

RANDOM

two things

what's up with the Django pony?

one

“Daddy! I want a pony!”

“No, you can’t have a pony.”

“Django, I want every feature!”

“No, you can’t have every feature.”

two

"I could take a framework seriously that had a mascot with magical powers!"

Cal Henderson - Djangocon 2008

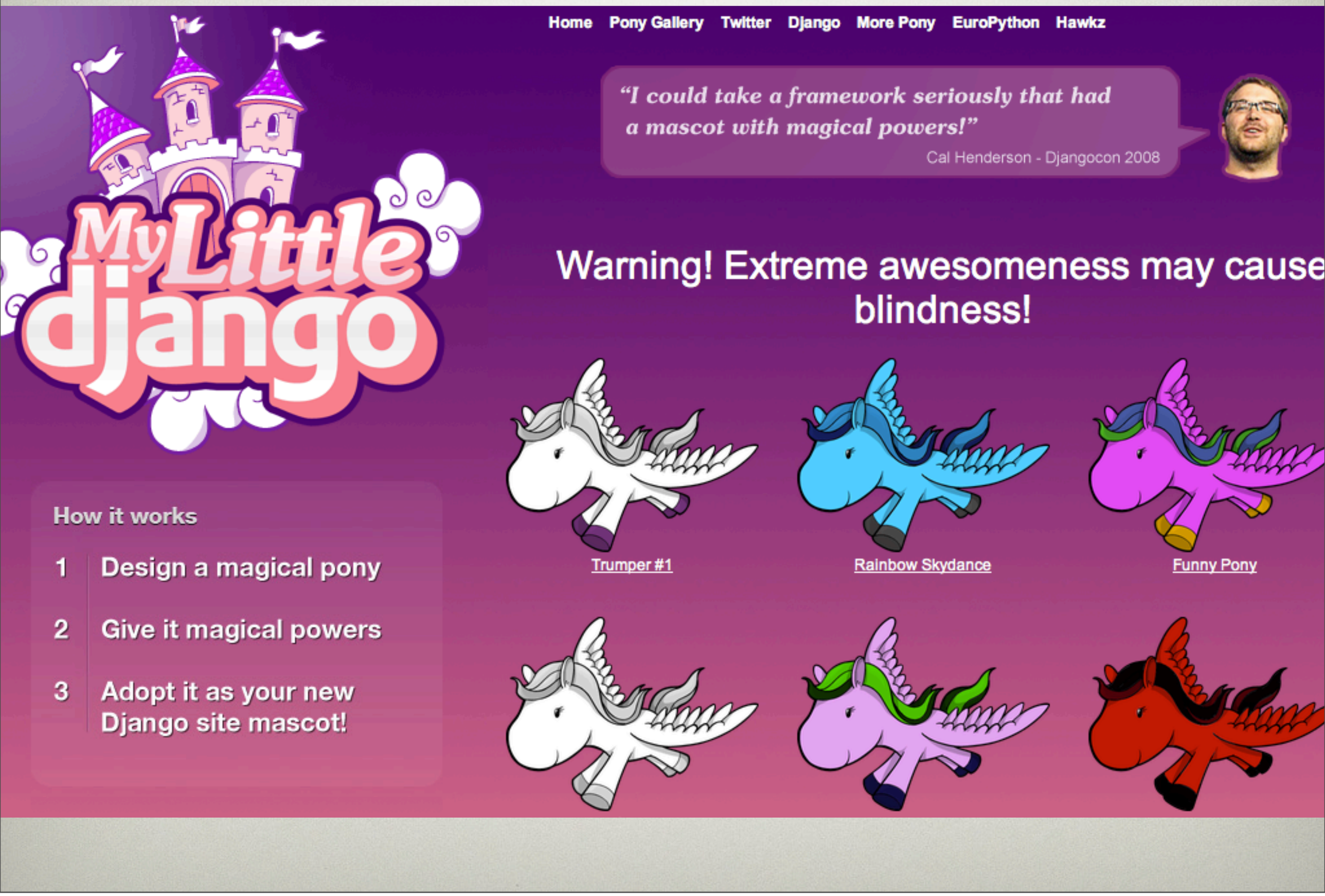

django

© Ted Leung

© Ted Leung

http://www.djangopony.com/

http://www.mylittledjango.com/



Ponies everywhere

Full story here:

http://avalonstar.com/blog/2008/sep/9/the-web-framework-for-ponies/

Lecture B (20)

documentation
&
getting help

watch your coding,
get in the habit of
checking the docs
when things get hard

If it seems like you're working pretty hard to make something happen, there is probably something built in to make it easier.

We've put a lot of effort into making Django's documentation useful, easy to read and as complete as possible. The rest of this document explains more about how the documentation works so that you can get the most out of it.

(Yes, this is documentation about documentation. Rest assured we have no plans to write a document about how to read the document about documentation.)

Finding documentation

Django's got a *lot* of documentation – almost 200,000 words – so finding what you need can sometimes be tricky. A few good places to start are the Search Page and the Index.

http://docs.djangoproject.com/en/1.2/intro/whatsnext/

This document is really good, read it.

Django documentation

Everything you need to know about Django (and then some).

First steps

- **From scratch:** Overview | Installation
- **Tutorial:** Part 1 | Part 2 | Part 3 | Part 4

The model layer

- **Models:** Model syntax | Field types | Meta options
- **QuerySets:** Executing queries | QuerySet method reference
- **Model instances:** Instance methods | Accessing related objects
- **Advanced:** Managers | Raw SQL | Transactions | Aggregation | Custom fields | Multiple databases
- **Other:** Supported databases | Legacy databases | Providing initial data | Optimize database access

The template layer

- **For designers:** Syntax overview | Built-in tags and filters
- **For programmers:** Template API | Custom tags and filters

The view layer

- **The basics:** URLconfs | View functions | Shortcuts | Decorators
- **Reference:** Request/response objects
- **File uploads:** Overview | File objects | Storage API | Managing files | Custom storage
- **Generic views:** Overview | Built-in generic views
- **Advanced:** Generating CSV | Generating PDF
- **Middleware:** Overview | Built-in middleware classes

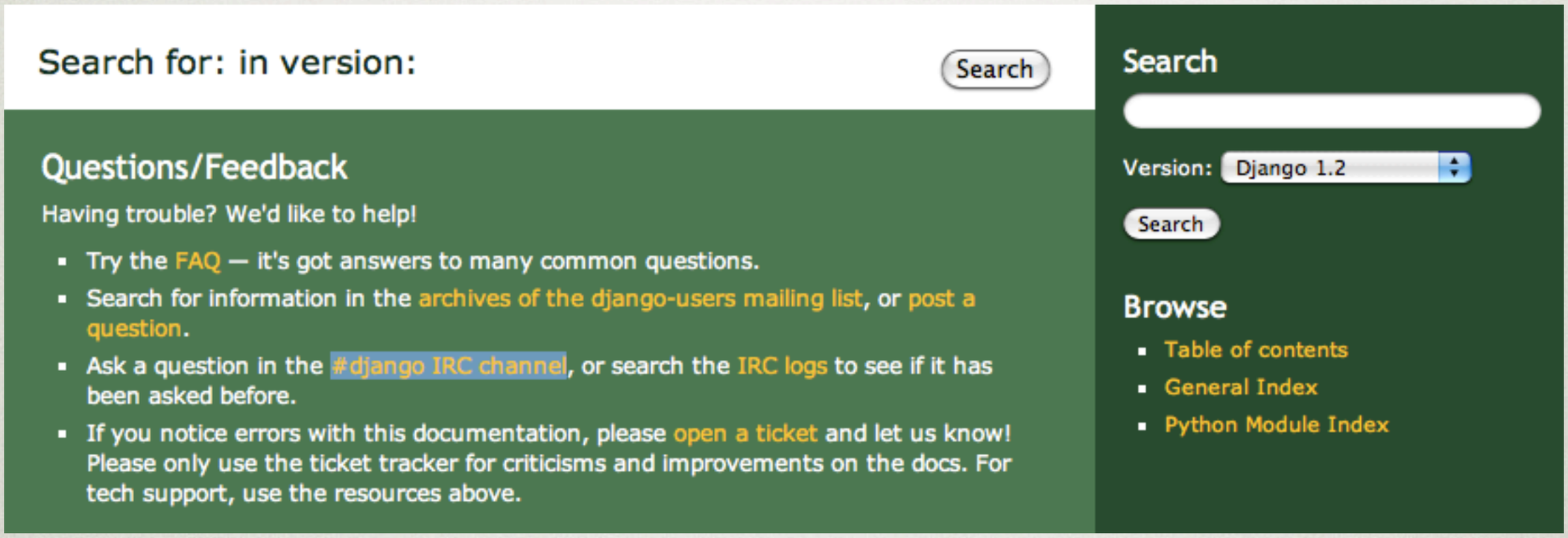
Getting help

Having trouble? We'd like to help!

- Try the FAQ – it's got answers to many common questions.
- Looking for specific information? Try the Index, Module Index or the detailed table of contents.
- Search for information in the archives of the django-users mailing list, or post a question.
- Ask a question in the #django IRC channel, or search the IRC logs to see if it's been asked before.
- Report bugs with Django in our ticket tracker.

http://docs.djangoproject.com/en/1.2/

search



http://docs.djangoproject.com/en/1.2/search/

index

General Index

Symbols | _ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

Symbols

--addrport
django-admin command-line option

--adminmedia
django-admin command-line option

--all
django-admin command-line option

--blank
django-admin command-line option

--database
django-admin command-line option

--decimal
django-admin command-line option

--domain
django-admin command-line option

--email
django-admin command-line option

--exclude
django-admin command-line option

--extension
django-admin command-line option

--failfast
django-admin command-line option

http://docs.djangoproject.com/en/1.2/genindex/

module index

Module Index

c | d | f | h | m | s | t | u | v

C

- django.conf.urls.defaults
- django.contrib.admin
  Django's admin site.
- django.contrib.admindocs
  Django's admin documentation generator.
- django.contrib.auth
  Django's authentication framework.
- django.contrib.auth.backends
  Django's built-in authentication backend classes.
- django.contrib.auth.forms
- django.contrib.auth.middleware
  Authentication middleware.
- django.contrib.comments
  Django's comment framework
- django.contrib.comments.forms
  Forms for dealing with the built-in comment model.
- django.contrib.comments.models
  The built-in comment models
- django.contrib.comments.moderation
  Support for automatic comment moderation.
- django.contrib.comments.signals
  Signals sent by the comment module.
- django.contrib.contenttypes
  Provides generic interface to installed models.

http://docs.djangoproject.com/en/1.2/py-modindex/

Django FAQ

- FAQ: General
  - Why does this project exist?
  - What does "Django" mean, and how do you pronounce it?
  - Is Django stable?
  - Does Django scale?
  - Who's behind this?
  - Which sites use Django?
  - Django appears to be a MVC framework, but you call the Controller the "view", and the View the "template". How come you don't use the standard names?
  - <Framework X> does <feature Y> – why doesn't Django?
  - Why did you write all of Django from scratch, instead of using other Python libraries?
  - Is Django a content-management-system (CMS)?
  - How can I download the Django documentation to read it offline?
  - Where can I find Django developers for hire?
- FAQ: Installation
  - How do I get started?
  - What are Django's prerequisites?

http://docs.djangoproject.com/en/1.2/faq/

Using Django

Introductions to all the key parts of Django you'll need to know:

- How to install Django
- Models and databases
- Handling HTTP requests
- Working with forms
- The Django template language
- Generic views
- Managing files
- Testing Django applications
- User authentication in Django
- Django's cache framework
- Conditional View Processing
- Sending e-mail
- Internationalization and localization
- Pagination
- Serializing Django objects
- Django settings
- Signals

mailing list

http://groups.google.com/group/django-users/

IRC

`#django -- irc://irc.freenode.net/`

Internet Relay Chat
Group IM
venerable internet protocol - predates the web

tons of blogs, books, etc

don't worry *too* much about the dates, Django is pretty stable - most things written in the last few years will be reasonably accurate

request

|

Django magic

|

response

Loose coupling

A fundamental goal of Django's stack is loose coupling and tight cohesion. The various layers of the framework shouldn't "know" about each other unless absolutely necessary.

For example, the template system knows nothing about Web requests, the database layer knows nothing about data display and the view system doesn't care which template system a programmer uses.

Although Django comes with a full stack for convenience, the pieces of the stack are independent of another wherever possible.

Less code

Django apps should use as little code as possible; they should lack boilerplate. Django should take full advantage of Python's dynamic capabilities, such as introspection.

http://docs.djangoproject.com/en/1.2/misc/design-philosophies/

Quick development

The point of a Web framework in the 21st century is to make the tedious aspects of Web development fast. Django should allow for incredibly quick Web development.

Don't repeat yourself (DRY)

Every distinct concept and/or piece of data should live in one, and only one, place. Redundancy is bad. Normalization is good.

The framework, within reason, should deduce as much as possible from as little as possible.

See also

The discussion of DRY on the Portland Pattern Repository

Explicit is better than implicit

This, a core Python principle, means Django shouldn't do too much "magic." Magic shouldn't happen unless there's a really good reason for it. Magic is worth using only if it creates a huge convenience unattainable in other ways, and it isn't implemented in a way that confuses developers who are trying to learn how to use the feature.

Consistency

The framework should be consistent at all levels. Consistency applies to everything from low-level (the Python coding style used) to high-level (the "experience" of using Django).

"Any sufficiently advanced technology is indistinguishable from magic"

-- Arthur C. Clarke

Magic is in the eye of the beholder?

can we change the “magic” part to...

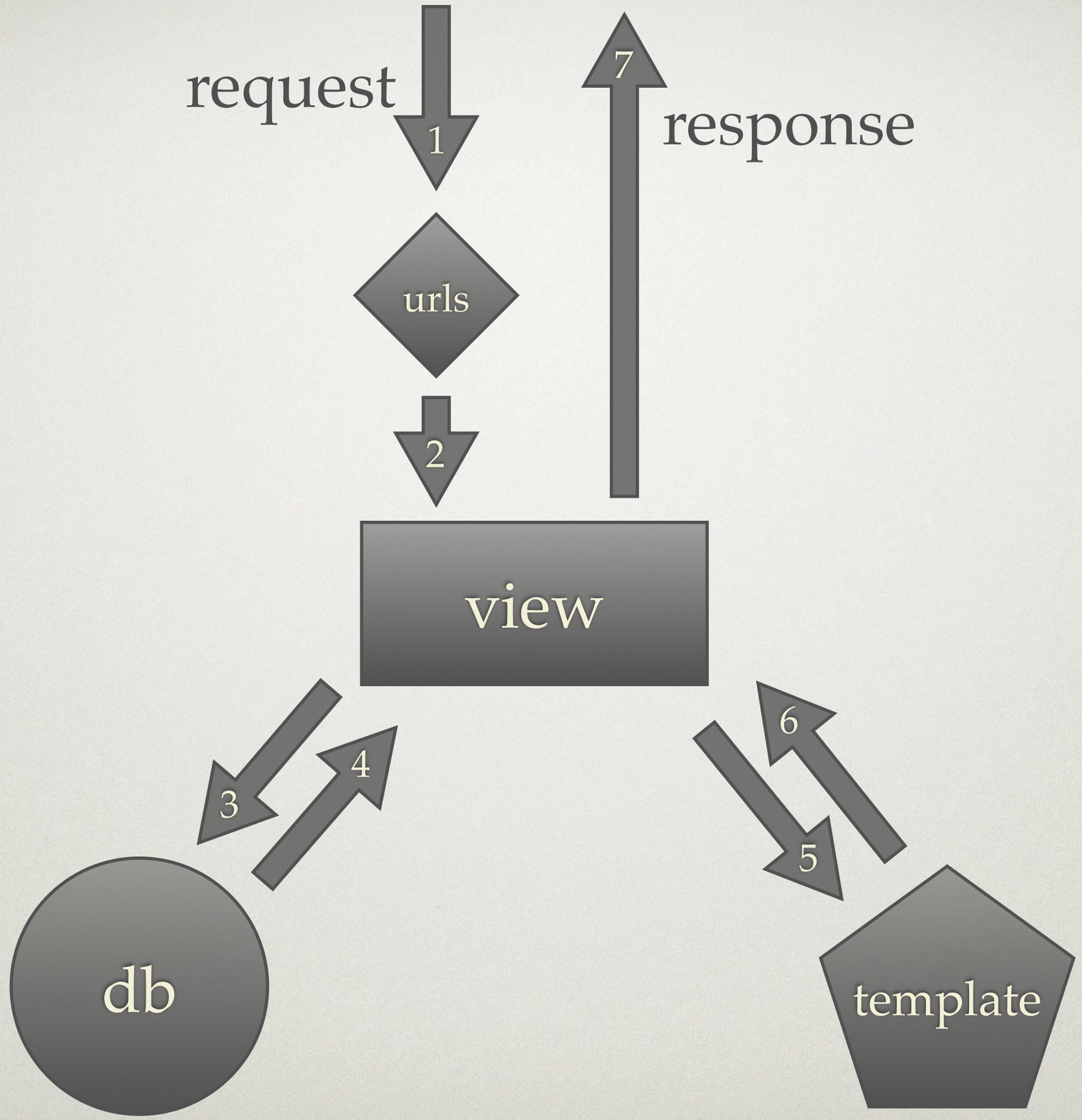
request

|

Django goodness

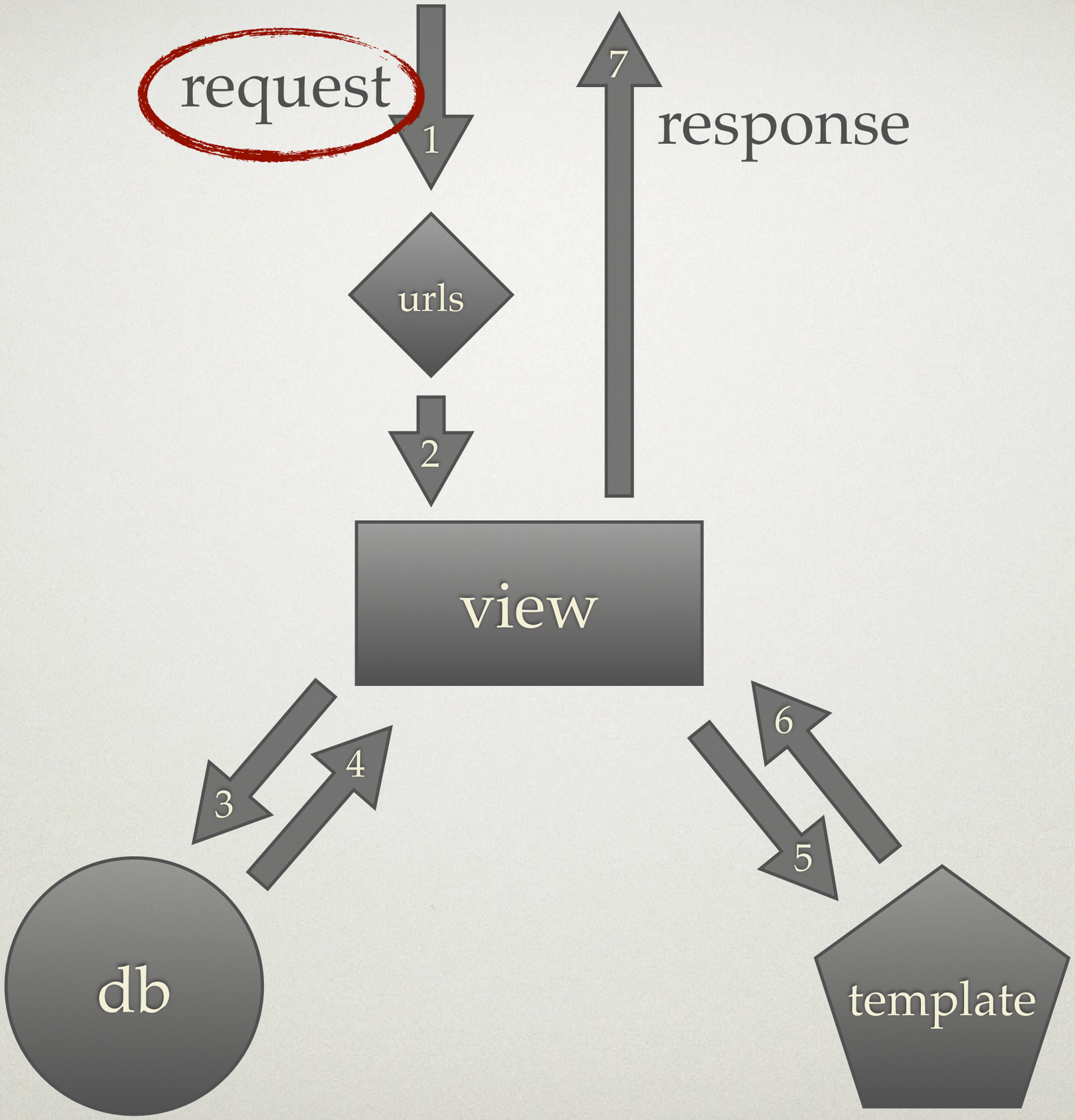
|

response

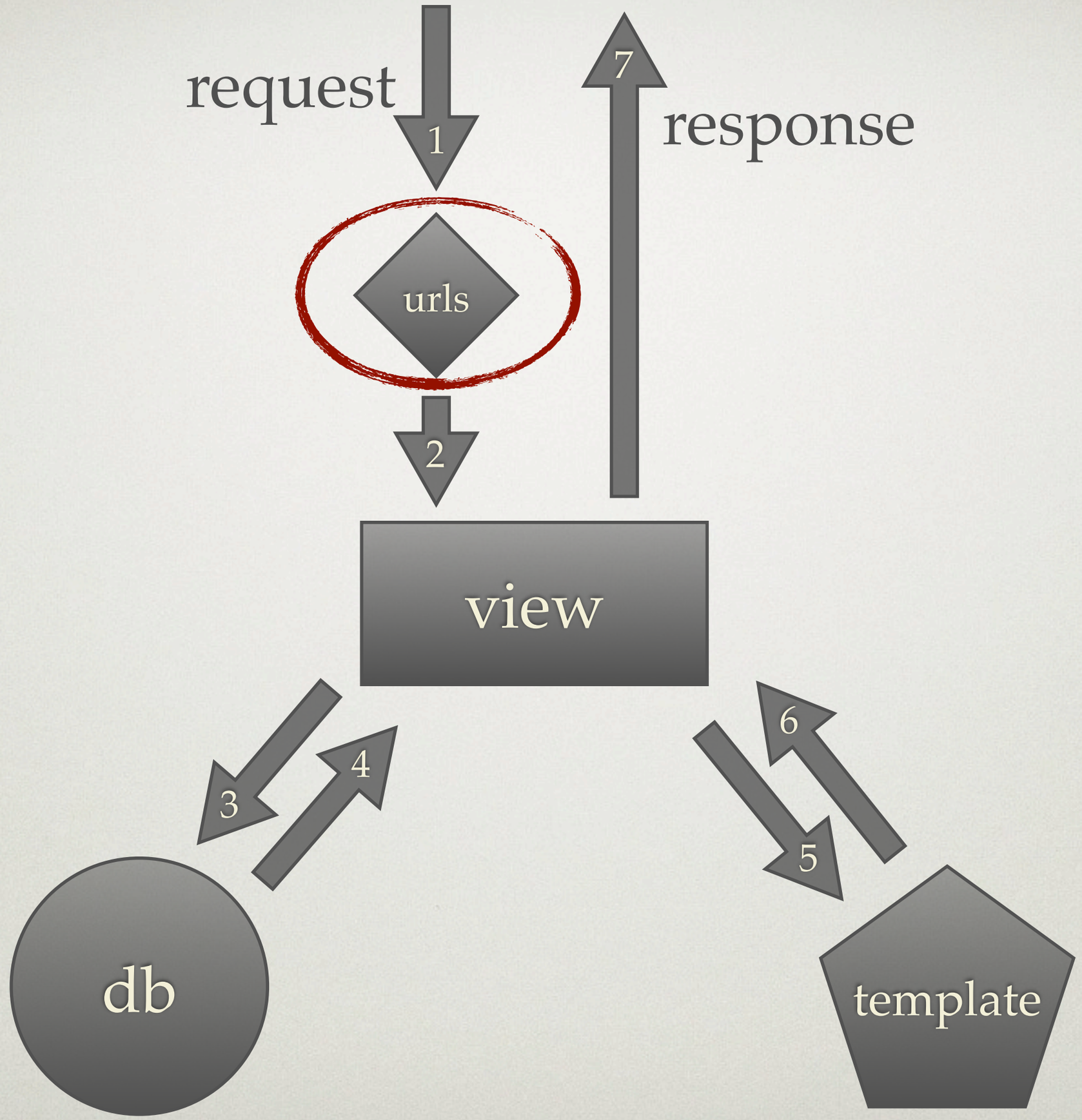


simplification

A common HTTP request-response path. Not all requests will go through all pieces. This is where we'll be spending the rest of the evening.



simplification



simplification

urls

URL design

Loose coupling

URLs in a Django app should not be coupled to the underlying Python code. Tying URLs to Python function names is a Bad And Ugly Thing.

Along these lines, the Django URL system should allow URLs for the same app to be different in different contexts. For example, one site may put stories at `/stories/`, while another may use `/news/`.

Infinite flexibility

URLs should be as flexible as possible. Any conceivable URL design should be allowed.

Encourage best practices

The framework should make it just as easy (or even easier) for a developer to design pretty URLs than ugly ones.

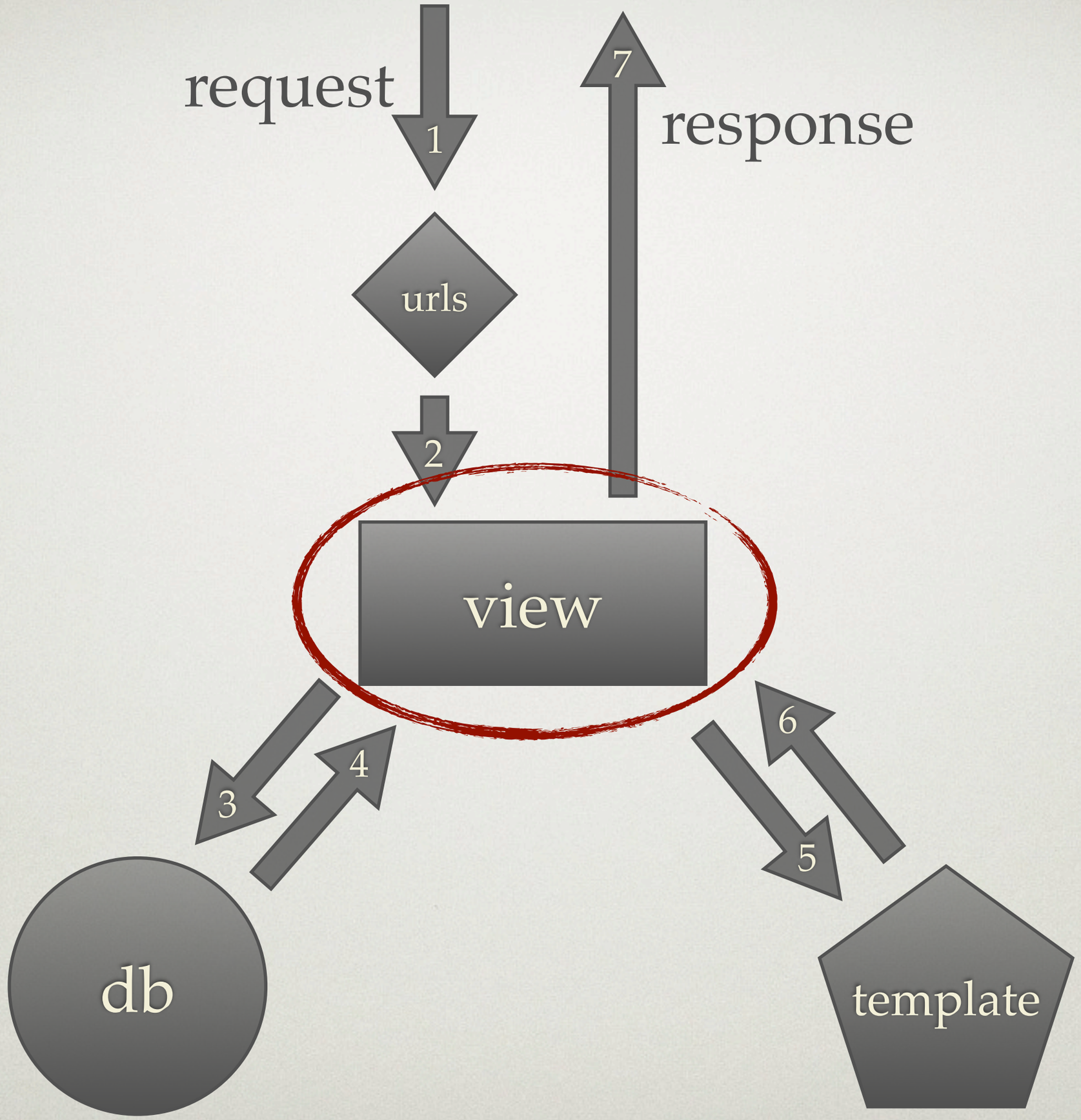
File extensions in Web-page URLs should be avoided.

Vignette-style commas in URLs deserve severe punishment.

Definitive URLs

Technically, `foo.com/bar` and `foo.com/bar/` are two different URLs, and search-engine robots (and some Web traffic-analyzing tools) would treat them as separate pages. Django should make an effort to "normalize" URLs so that search-engine robots don't get confused.

This is the reasoning behind the `APPEND_SLASH` setting.



simplification

views

Views

Simplicity

Writing a view should be as simple as writing a Python function. Developers shouldn't have to instantiate a class when a function will do.

Use request objects

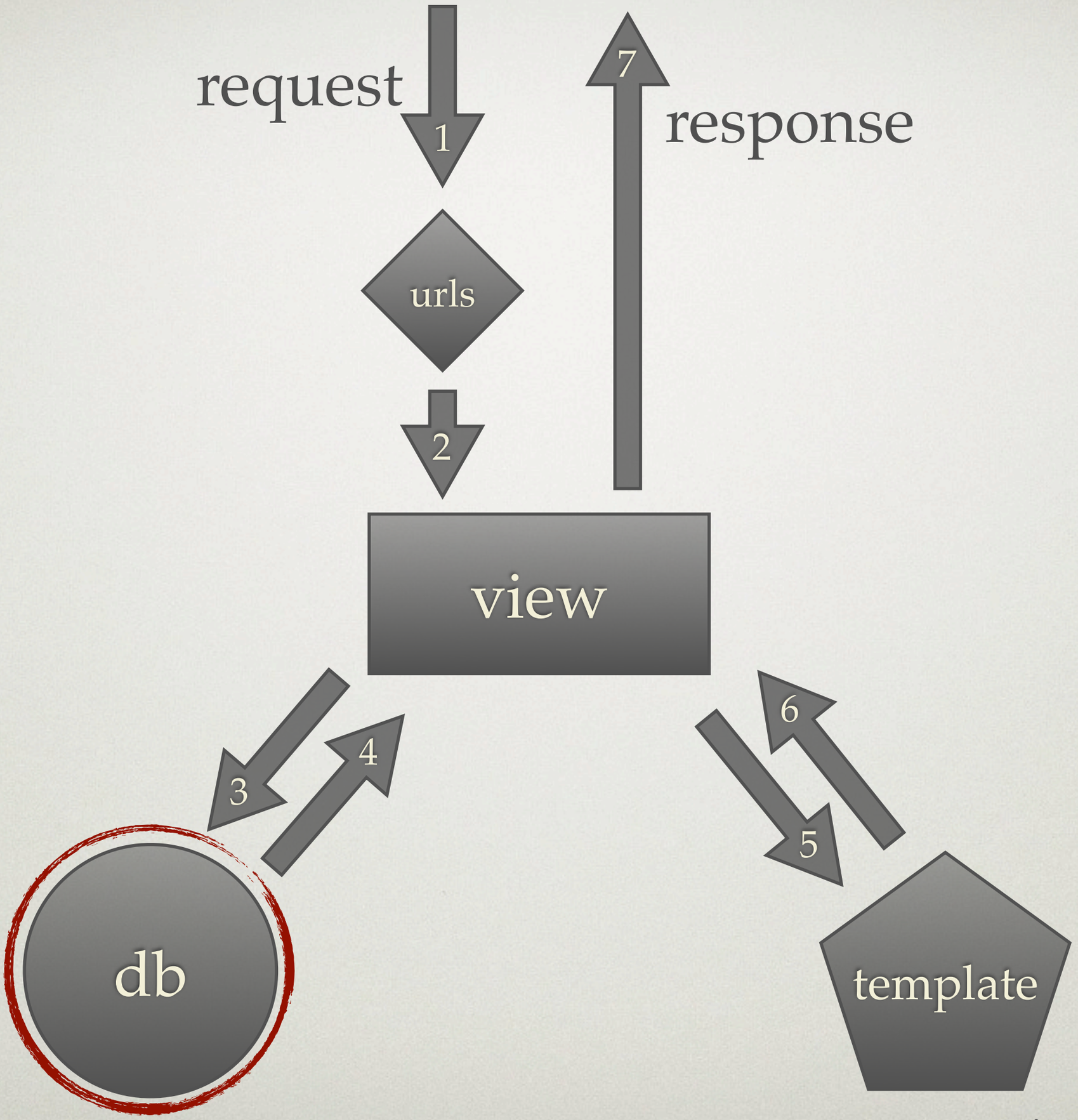
Views should have access to a request object – an object that stores metadata about the current request. The object should be passed directly to a view function, rather than the view function having to access the request data from a global variable. This makes it light, clean and easy to test views by passing in "fake" request objects.

Loose coupling

A view shouldn't care about which template system the developer uses – or even whether a template system is used at all.

Differentiate between GET and POST

GET and POST are distinct; developers should explicitly use one or the other. The framework should make it easy to distinguish between GET and POST data.



simplification

models

(a digression)

Models

Explicit is better than implicit

Fields shouldn't assume certain behaviors based solely on the name of the field. This requires too much knowledge of the system and is prone to errors. Instead, behaviors should be based on keyword arguments and, in some cases, on the type of the field.

Include all relevant domain logic

Models should encapsulate every aspect of an "object," following Martin Fowler's Active Record design pattern.

This is why both the data represented by a model and information about it (its human-readable name, options like default ordering, etc.) are defined in the model class; all the information needed to understand a given model should be stored *in* the model.

also: DRY

model api

Database API

The core goals of the database API are:

SQL efficiency

It should execute SQL statements as few times as possible, and it should optimize statements internally.

This is why developers need to call `save()` explicitly, rather than the framework saving things behind the scenes silently.

This is also why the `select_related()` `QuerySet` method exists. It's an optional performance booster for the common case of selecting "every related object."

Terse, powerful syntax

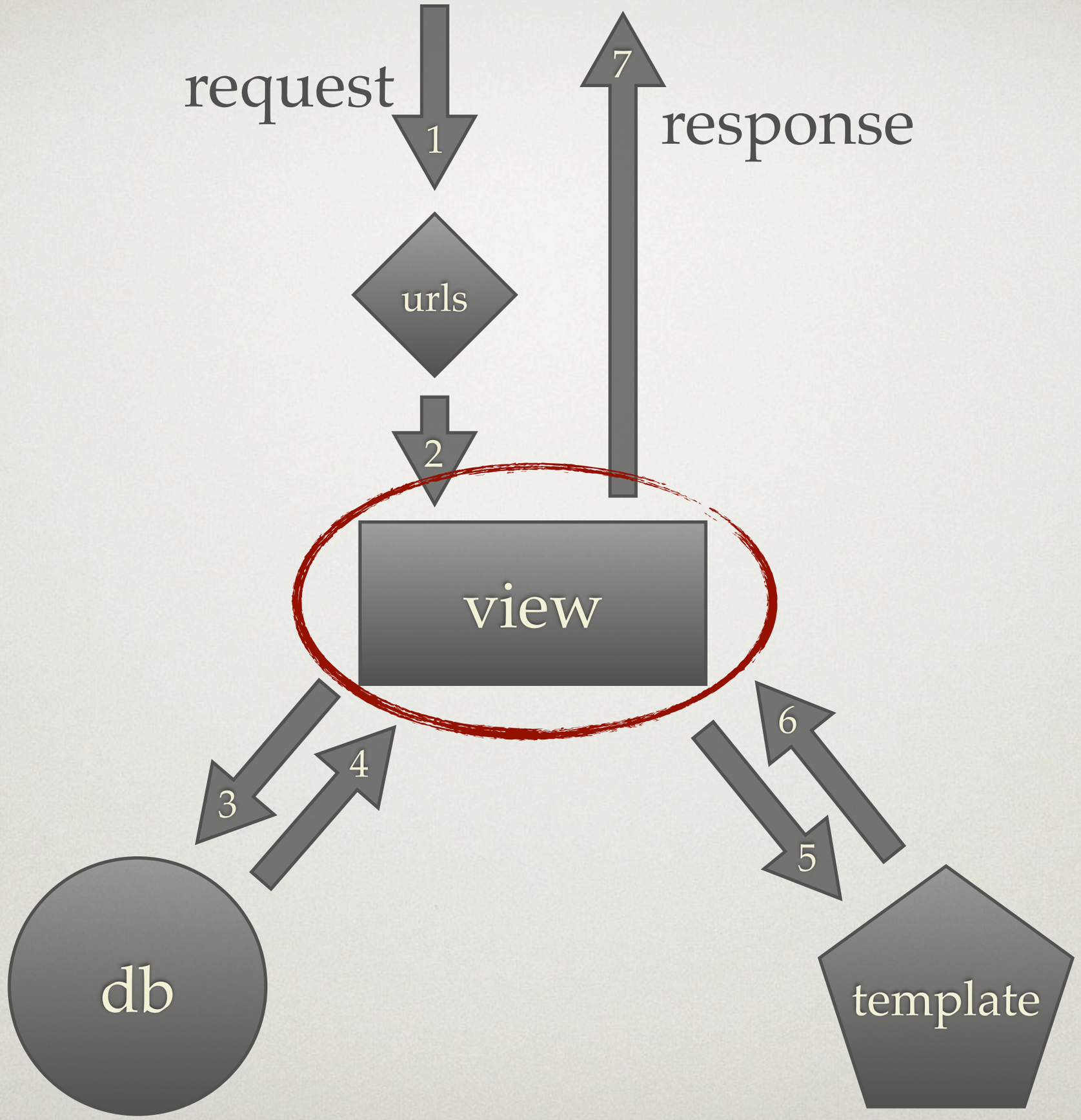
The database API should allow rich, expressive statements in as little syntax as possible. It should not rely on importing other modules or helper objects.

Joins should be performed automatically, behind the scenes, when necessary.

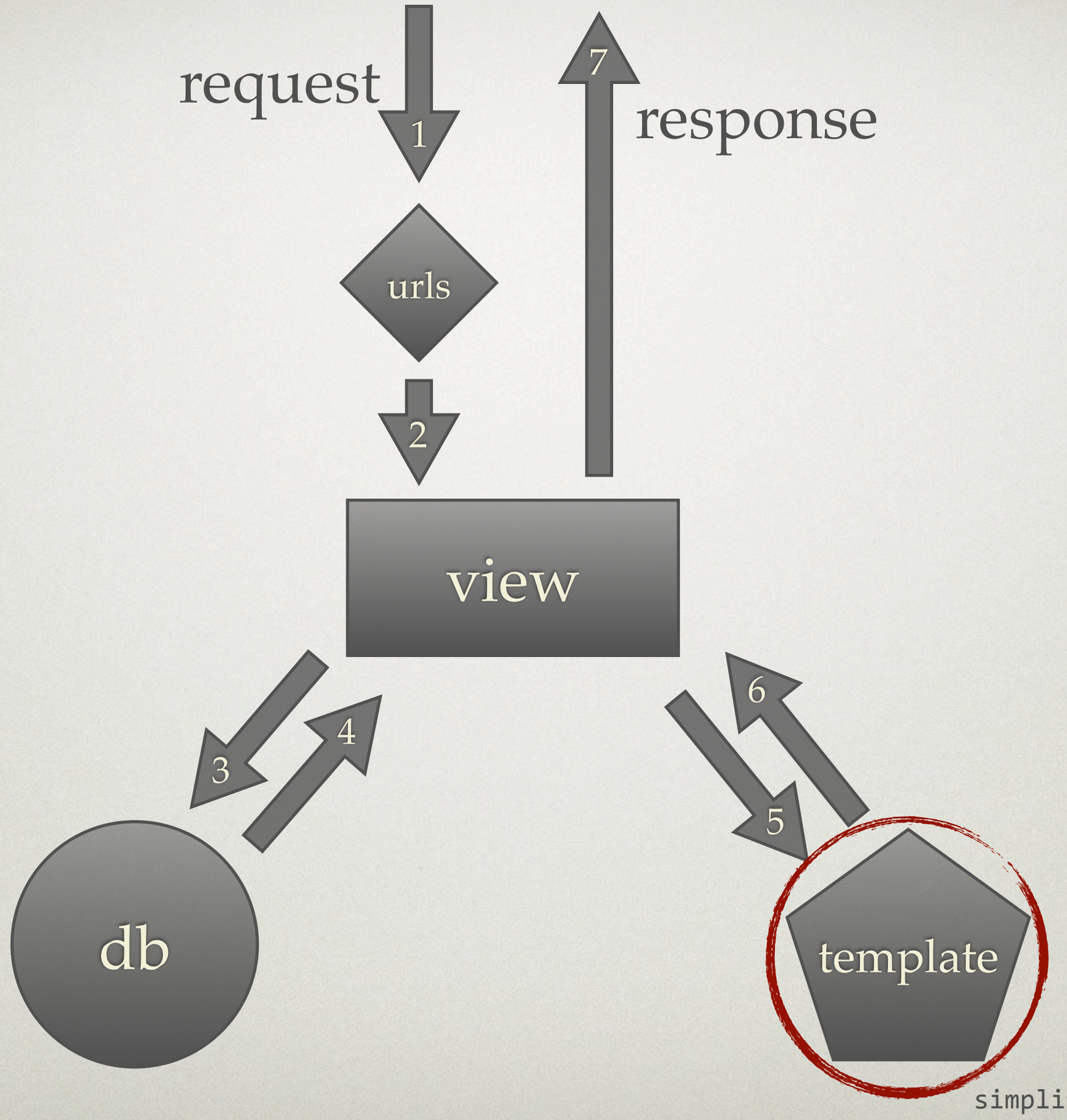
Every object should be able to access every related object, systemwide. This access should work both ways.

Option to drop into raw SQL easily, when needed

The database API should realize it's a shortcut but not necessarily an end-all-be-all. The framework should make it easy to write custom SQL – entire statements, or just custom `WHERE` clauses as custom parameters to API calls.



simplification



simplification

templates

Template system

Separate logic from presentation

We see a template system as a tool that controls presentation and presentation-related logic – and that's it. The template system shouldn't support functionality that goes beyond this basic goal.

If we wanted to put everything in templates, we'd be using PHP. Been there, done that, wised up.

Discourage redundancy

The majority of dynamic Web sites use some sort of common sitewide design – a common header, footer, navigation bar, etc. The Django template system should make it easy to store those elements in a single place, eliminating duplicate code.

This is the philosophy behind template inheritance.

Be decoupled from HTML

The template system shouldn't be designed so that it only outputs HTML. It should be equally good at generating other text-based formats, or just plain text.

XML should not be used for template languages

Using an XML engine to parse templates introduces a whole new world of human error in editing templates – and incurs an unacceptable level of overhead in template processing.

Assume designer competence

The template system shouldn't be designed so that templates necessarily are displayed nicely in WYSIWYG editors such as Dreamweaver. That is too severe of a limitation and wouldn't allow the syntax to be as nice as it is. Django expects template authors are comfortable editing HTML directly.

Treat whitespace obviously

The template system shouldn't do magic things with whitespace. If a template includes whitespace, the system should treat the whitespace as it treats text – just display it. Any whitespace that's not in a template tag should be displayed.

Don't invent a programming language

The template system intentionally doesn't allow the following:

- Assignment to variables
- Advanced logic

The goal is not to invent a programming language. The goal is to offer just enough programming-esque functionality, such as branching and looping, that is essential for making presentation-related decisions.

The Django template system recognizes that templates are most often written by *designers*, not *programmers*, and therefore should not assume Python knowledge.

Safety and security

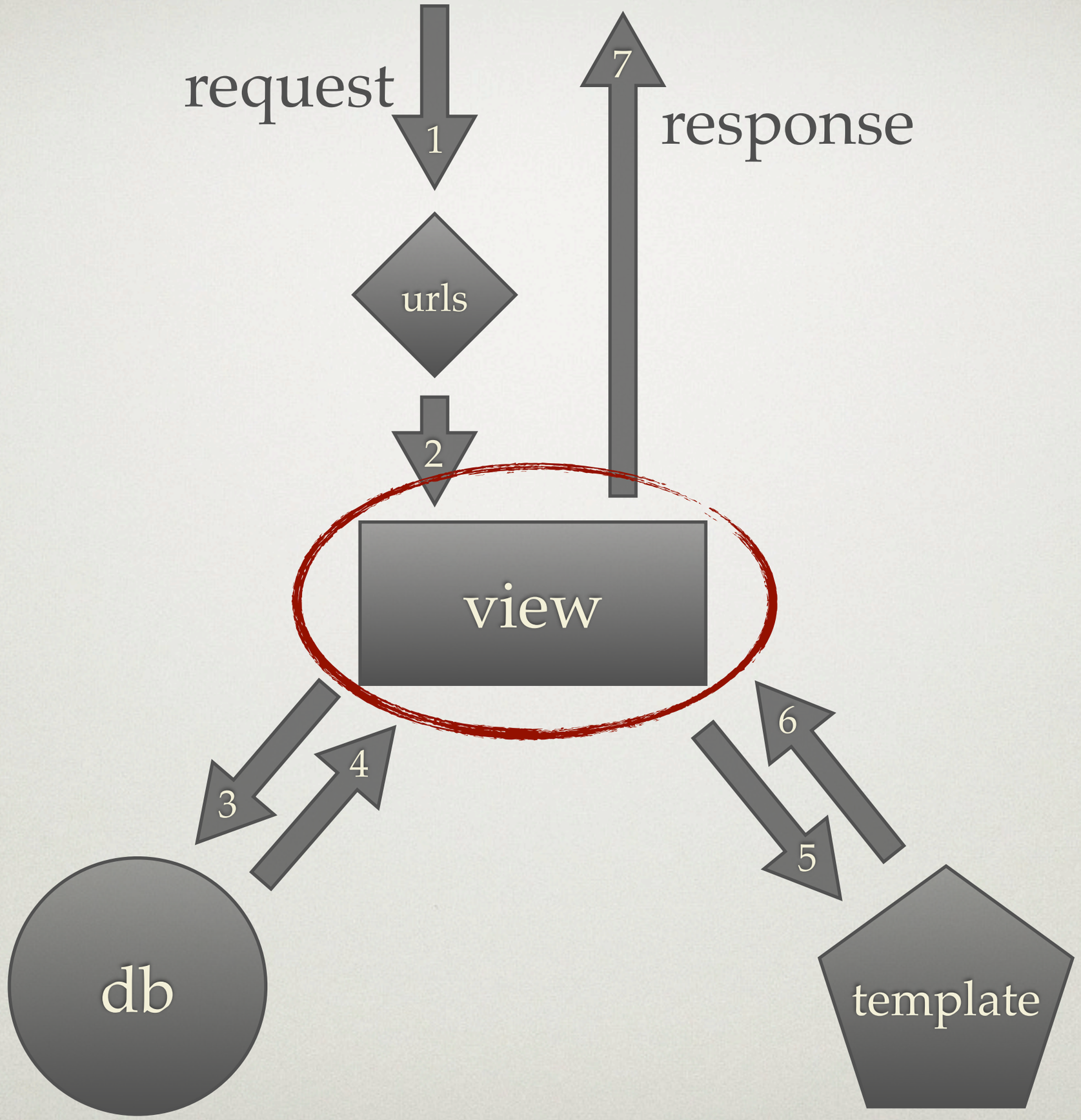
The template system, out of the box, should forbid the inclusion of malicious code – such as commands that delete database records.

This is another reason the template system doesn't allow arbitrary Python code.

Extensibility

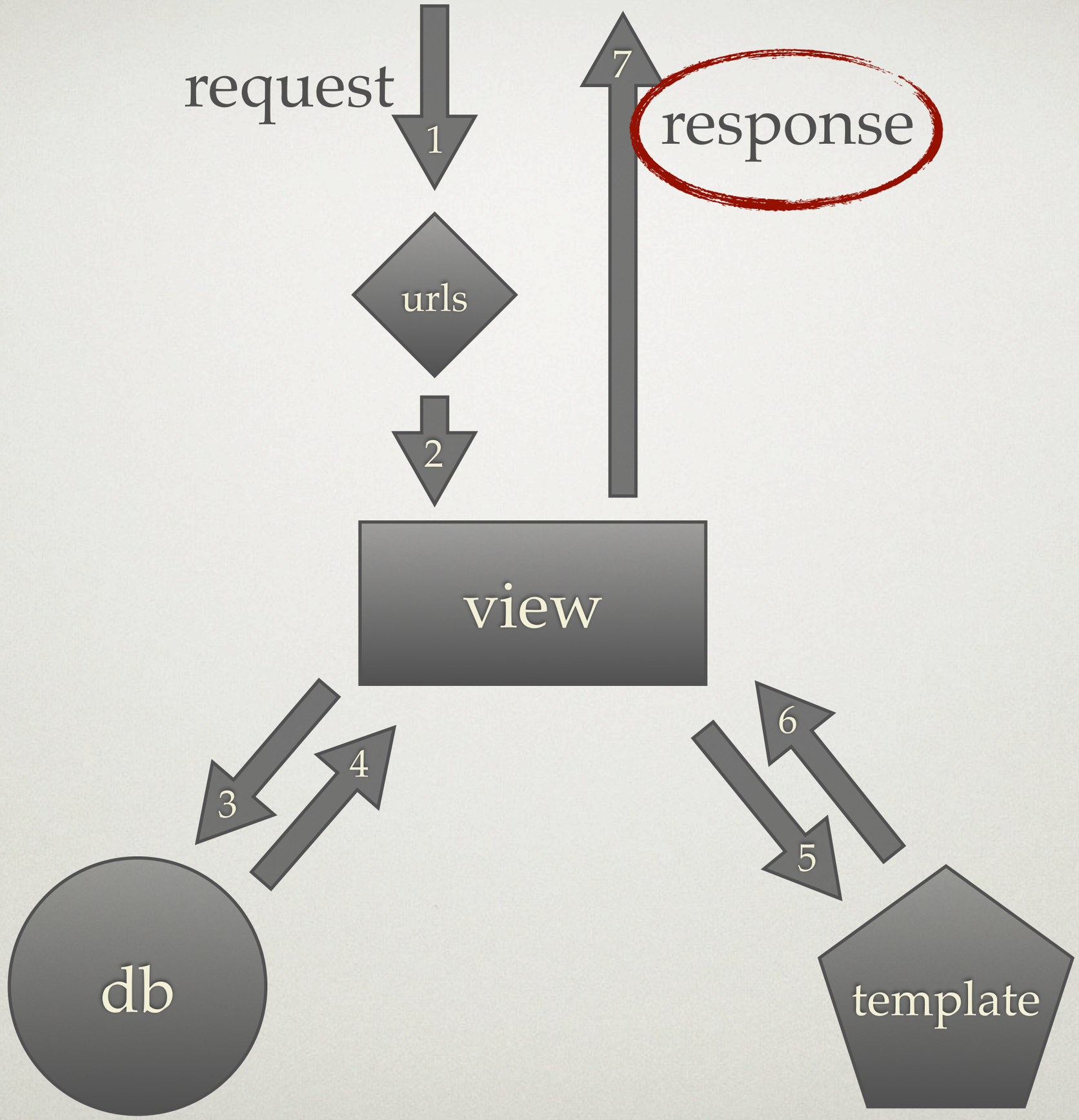
The template system should recognize that advanced template authors may want to extend its technology.

This is the philosophy behind custom template tags and filters.



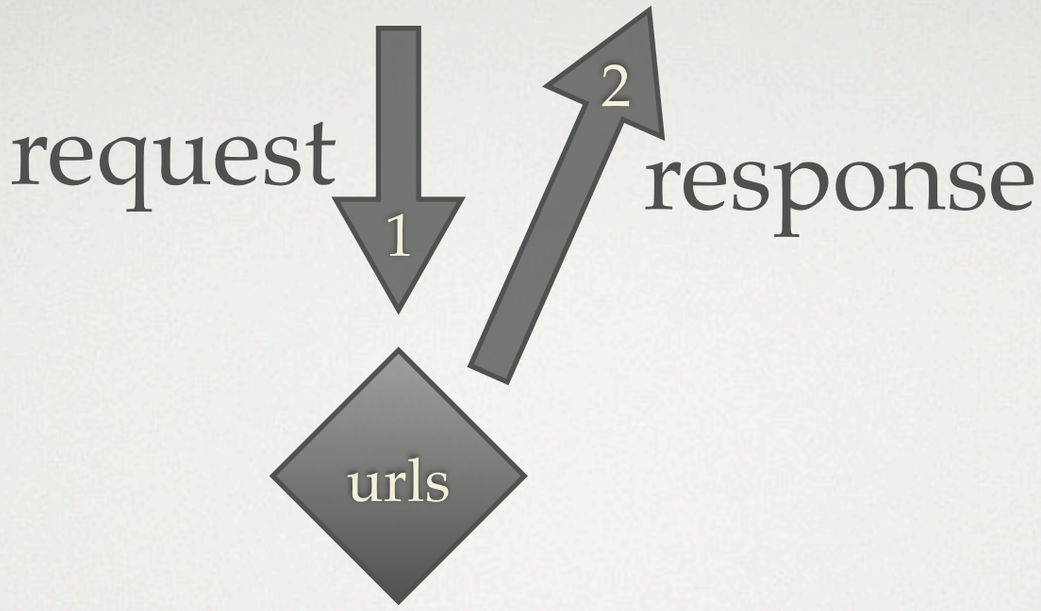
simplification

success!

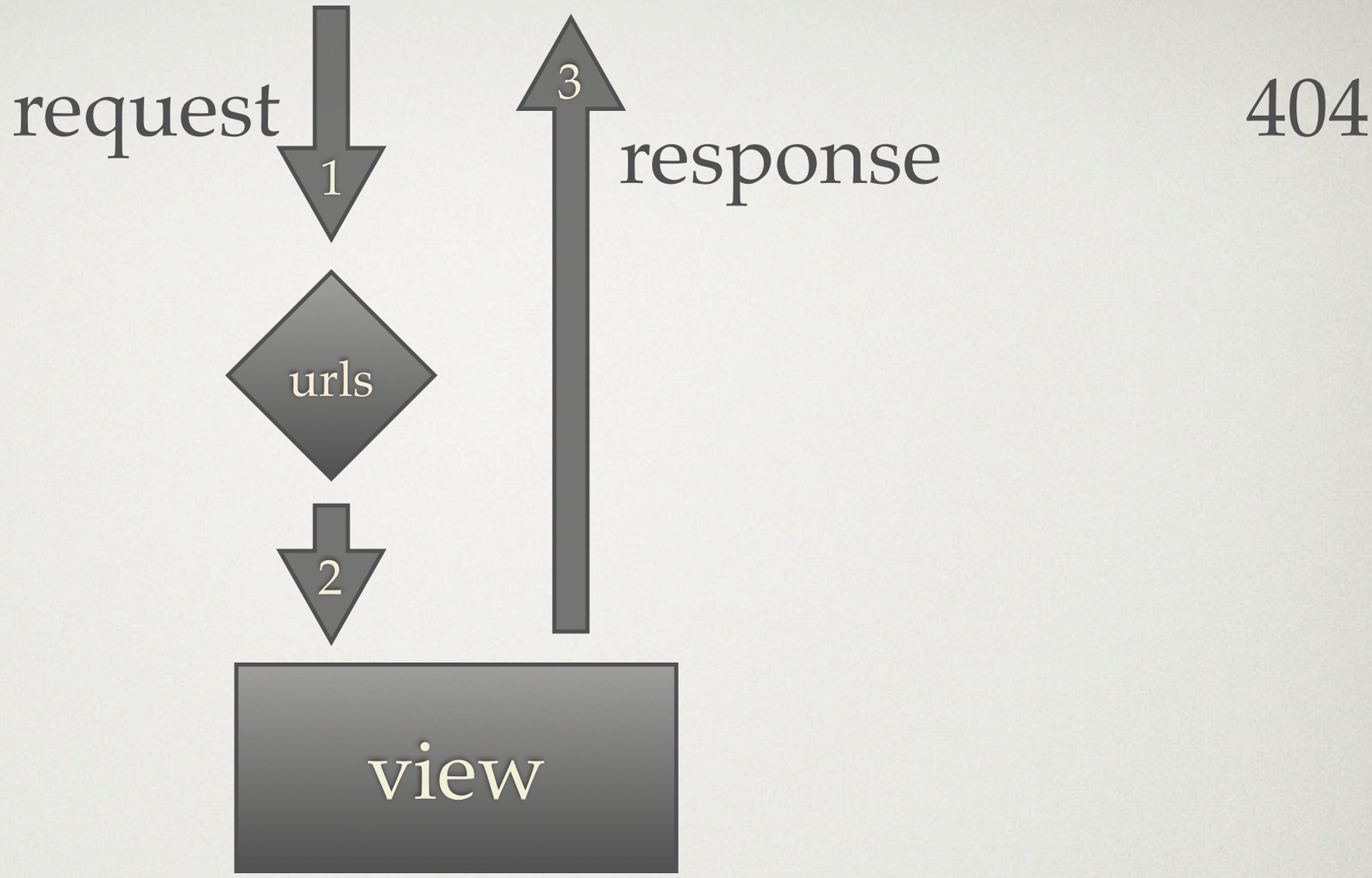


simplification

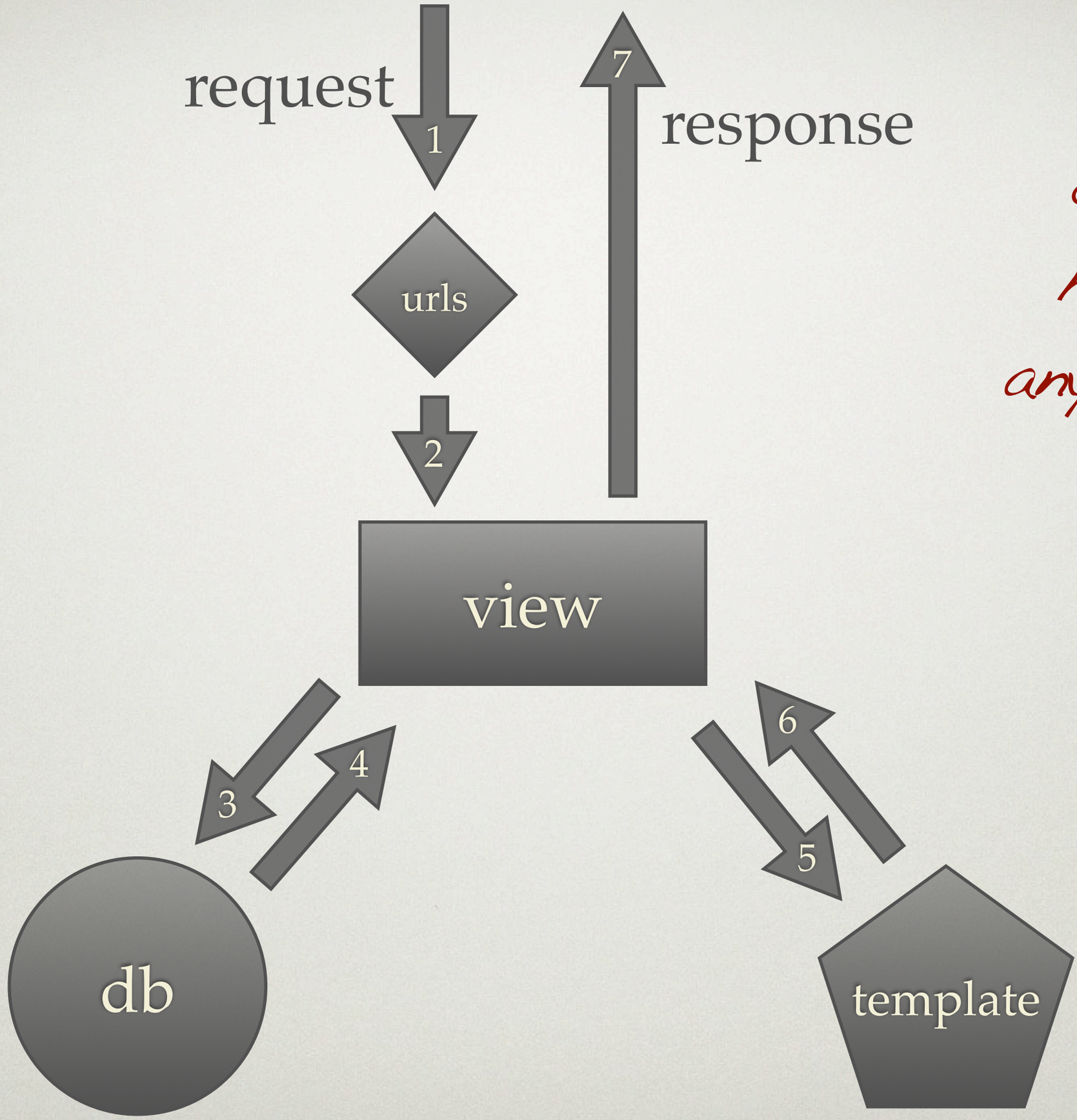
some other paths



simplification



simplification



500

could happen anywhere!

simplification

any uncaught Python exception

admin UI

next week

- project layout
  - apps vs projects
  - on disk organization
- more admin customization
- django.contrib
  - optional, de facto standard implementations of common patterns
- urls
  - named urls
  - reverse(), permalink, {% url %}
- database API
  - related objects
  - QuerySets
  - raw SQL
  - non-relational DBs
- templates
  - overview
  - template inheritance
  - custom template tags
- caching
- forms
  - overview
  - API
  - fields and widgets
  - model based forms
- signals
- unicode & encodings
  - defaults
  - how to change it
- middleware
- good stuff from django-admin and manage
- a taste of testing
  - unit test
  - Django extras
  - fixtures
- syndication? (rss, etc)
- extra
  - geodjango
  - stand alone scripts which use the database API
- other people's apps
  - where to find them
  - which ones should we look at first?
- deploying django
  - webfaction setup demo
- Django 1.3
  - what's new
  - what's being removed

(pause)

reflect
(on the format)

Lab B

(20)

2:35

Lab B

- Decide on your problem domain for this week's assignment

- Write the smallest models which could work

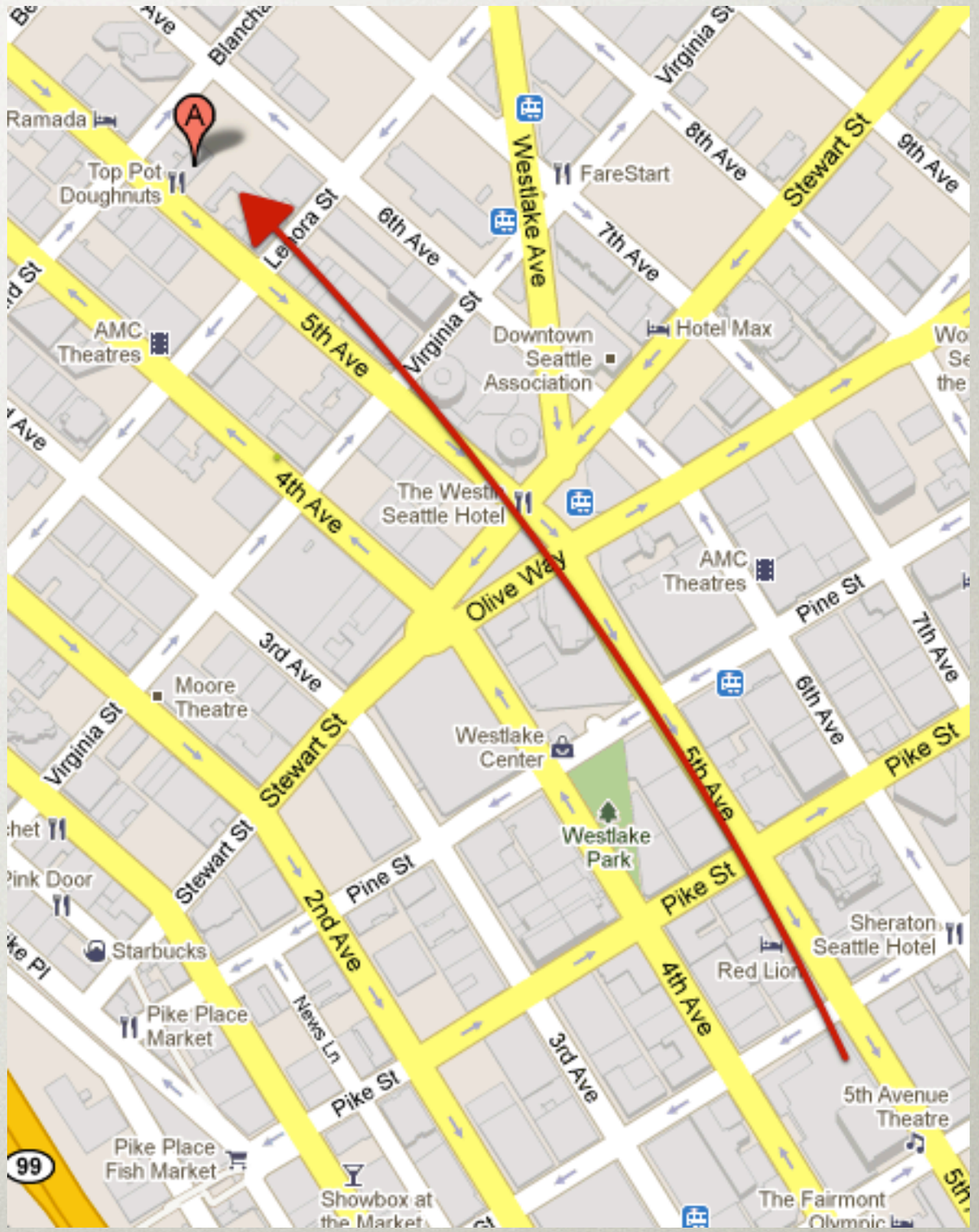
WRAPUP

2:55

Info

Office hours:
Sunday 2-5pm

Top Pot Donuts
2124 5th Ave
Seattle, WA 98121
206-728-1966



turn in here: `http://is.gd/uwipip_week7`

Assignment

- build a small website for something you (or your friends/family) collect or love
- admin UI is enough for editing
- list of all items
- item detail pages
- pivot on at least one item detail
  (ex: for books, click author, get list of books by that author)

pick at least one

- stretch: configure the admin UI to make it really clean
- stretch: searching
- stretch: items are owned, and can be loaned to other people.
- stretch: make it look decent - use design templates if needed